Scheduling Data Flow Program in XKaapi: A New Affinity Based Algorithm for Heterogeneous Architectures

Raphaël Bleuse, Thierry Gautier, João V. F. Lima, Grégory Mounié, Denis Trystram

Scheduling data flow program in XKaapi: A new affinity based Algorithm for Heterogeneous Architectures

Raphaël Bleuse[1], Thierry Gautier[2], João V. F. Lima[4], Grégory Mounié[1], and Denis Trystram[13]

{raphael.bleuse, gregory.mounie, denis.trystram}@imag.fr,
thierry.gautier@inrialpes.fr, jvlima@inf.ufsm.br

[1] Univ. Grenoble Alpes, France
[2] Inria Rhône-Alpes, France
[3] Institut universitaire de France
[4] Universidade Federal de Santa Maria (UFSM), Brazil

Abstract. Efficient implementations of parallel applications on heterogeneous hybrid architectures require a careful balance between computations and communications with accelerator devices. Even if most of the communication time can be overlapped by computations, it is essential to reduce the total volume of communicated data. The literature therefore abounds with *ad hoc* methods to reach that balance, but these are architecture and application dependent. We propose here a generic mechanism to automatically optimize the scheduling between CPUs and GPUs, and compare two strategies within this mechanism: the classical Heterogeneous Earliest Finish Time (HEFT) algorithm and our new, parametrized, Distributed Affinity Dual Approximation algorithm (DADA), which consists in grouping the tasks by affinity before running a fast dual approximation. We ran experiments on a heterogeneous parallel machine with twelve CPU cores and eight NVIDIA Fermi GPUs. Three standard dense linear algebra kernels from the PLASMA library have been ported on top of the XKaapi runtime system. We report their performances. It results that HEFT and DADA perform well for various experimental conditions, but that DADA performs better for larger systems and number of GPUs, and, in most cases, generates much lower data transfers than HEFT to achieve the same performance.

Keywords: heterogeneous architectures, scheduling, cost models, dual approximation scheme, programming tools, affinity

1 Introduction

With the recent evolution of processor design, the future generations of processors will contain hundreds of cores. To increase the performance per watt ratio, the cores will be non-symmetric with few highly powerful cores (CPU) and numerous, but simpler, cores (GPU). The success of such machines will rely on the ability to schedule the workload at runtime, even for small problem instances.

One of the main challenges is to define a scheduling strategy that may be able to exploit all potential parallelisms on a heterogeneous architecture composed of multiple CPUs and multiple GPUs. Previous works demonstrate the efficiency of strategies such as static distribution [14, 15], centralized list scheduling with data locality [6], cost models [1–4] based on Heterogeneous-Earliest-Finish-Time scheduling (HEFT) [16], and dynamic for some specific application domains [5,10]. Locality-aware work stealing [9], with a careful implementation to overlap communication by computation [13], improves significantly the performance of compute-bound linear algebra problems such as matrix product and Cholesky factorization.

Nevertheless, none of the above cited works considers scheduling strategies from the viewpoint of a compromise between performance and locality. In this paper, we propose a scheduling algorithm based on dual approximation [12] that uses a performance model to predict the execution time of tasks during scheduling decision. This algorithm, called Distributed Affinity Dual Approximation (DADA), is able to find a compromise between transfers and performance. It is parametrized by α for tuning this trade-off. The main advantage of dual approximation algorithms is their theoretical performance guarantee as they have a constant approximation ratio. On the contrary, the worst case of HEFT can be arbitrarily bad [12].

We compare these two different scheduling strategies for data-flow task programming. These strategies are implemented on top of the XKaapi scheduling framework with performance models for task execution time and transfer prediction. The contributions of this paper are first the design and implementation of dual approximation scheduling algorithms (with and without affinity) and second their evaluation in comparison to the well-known HEFT algorithm on three dense linear algebra algorithms in double precision floating-point operations from PLASMA [7]: namely Cholesky, LU, and QR. To our knowledge, this paper is the first report of experimental evaluations studying the impact of data transfer model and contention on a machine with up to 8 GPUs.

The main lesson of this work is that scheduling algorithms need extra information in order to take the right decisions. Such extra information could be obtained in a precise communication model to predict processing time of each task or in a more flexible information such as the affinity in DADA. Even if HEFT remains a good candidate for scheduling such linear algebra kernels, DADA is highly competitive against it for multi-GPU systems: the experimental results demonstrate that it achieves the same range of performances while reducing significantly the communication volume.

The remainder of this paper is organized as follows. Section 2 provides an overview of XKaapi runtime system, describes the XKaapi scheduling framework and the cost model applied for performance prediction. Section 3 details the two studied scheduling strategies. Section 4 presents our experimental results on a heterogeneous architecture composed of 12 CPUs and 8 GPUs. In Section 5 we briefly survey related works on runtime systems, scheduling strategies and

performance prediction. Finally, Section 6 concludes the paper and suggests future directions.

2 Scheduling framework in XKaapi

The XKaapi[5] data-flow model [8] – as in Cilk, Intel TBB, OpenMP-3.0, or OmpSs [6] – enables non-blocking task creation: the caller creates the task and proceeds with the program execution. Parallelism is explicit while the detection of synchronizations is implicit [8]: dependencies between tasks and memory transfers are automatically managed by the runtime system.

XKaapi runtime system is structured around the notion of *worker*: it is the internal representation of kernel threads. It executes the code of the tasks and takes local scheduling decisions. Each *worker* owns a local queue of ready tasks. Our interface is mainly inspired by work stealing scheduler and composed of three operations that act on workers' queues of tasks: *pop*, *push* and *steal*. In our previous work, we demonstrated the efficiency of XKaapi locality-aware work stealing as well as the corresponding multi-GPU runtime support [9] using specialized implementation of these operations. A new operation, called *activate*, has been defined to push ready task to a worker's queue.

2.1 Execution flow

The sketch of the execution mechanism is the following: at each step, either the own queue of worker is not empty and the worker uses it; or the worker emits a steal request to a randomly selected worker in order to get a task to execute. According to the dependencies between tasks, once a worker performs a task, it calls the *activate* operations in order to activate the successors of the task which become ready for execution.

The XKaapi runtime system gets information from each internal events (such as start-end of task execution, or start-end of communication toward GPU) to calibrate the performance model and corrects erroneous predictions due to unpredictable or unknown behavior (*e.g.* operating system state or I/O disturbance). StarPU [4] uses similar runtime measurements in order to correct the performance predictions in their HEFT implementation.

All of our scheduling strategies follow this sketch. Every worker terminates its execution when all the tasks of the application have been executed.

2.2 Pop, Push, Steal and Activate Operations

A framework interface for scheduling strategies is not a new concept in heterogeneous systems. Bueno *et al.* [6] and Augonnet *et al.* [4] described a minimal interface to design scheduling strategies with selection at runtime. However, there is little information available on the comparison of different strategies. Most of

[5] http://kaapi.gforge.inria.fr

them reported performance on centralized list scheduling and performance models. Our framework is composed of three classical operations in the work stealing context, plus an action to activate tasks when their predecessors have completed.

- The *push* operation inserts a task into a queue. A worker can push a task into any other workers' queue.
- A *pop* removes a task from the local queue owned by the caller worker.
- A *steal* removes a task from the queue of a remote worker. It is called by an idle thread – the *thief* – in order to pick tasks from a randomly selected worker – the *victim*.
- The *activate* operation is called after the completion of a task. The role of this operation is to allocate the tasks that are ready to be executed. Hence, most of the scheduling decision are done during this operation.

2.3 Performance Model

Cost models depend on a certain knowledge of both application algorithm and the underlying architecture to predict performance at runtime. In order to predict performance, we designed a StarPU [3] like performance model for task execution time and communication. Our task prediction relies on an history-based model, and transfer time estimation is based on asymptotic bandwidth. They are associated with scheduling strategies that are based on task completion time such as HEFT and DADA with and without affinity.

In order to balance efficiently the load, for each processor XKaapi maintains a shared time-stamp of the predicted time when it has completed its tasks. The completion date of the last executed task is also kept. The update and incrementation of the time-stamps are efficiently implemented with atomic operators.

3 Scheduling Strategies

This section introduces the scheduling strategies designed on top of the XKaapi scheduling framework. We consider a multi-core parallel architecture with m homogeneous CPUs and k homogeneous GPUs. First, we describe our implementation of HEFT [16]. Then, we recall the principle of the dual approximation scheme [11]. We propose a new algorithm – Distributed Affinity Dual Approximation (DADA) – based on this paradigm which takes into account the affinity between tasks.

In the following we denote by p_i^{CPU} the processing time of task T_i on a CPU and p_i^{GPU} on a GPU. We define the speedup S_i of task T_i as the ratio $S_i = p_i^{CPU} / p_i^{GPU}$.

3.1 HEFT within XKaapi

The Heterogeneous Earliest-Finish-Time algorithm (HEFT), proposed by [16], is a scheduling algorithm for a bounded number of heterogeneous processors. Its

Algorithm 1: HEFT – *activate* operation.

Input : A list of ready tasks T_i LR
Output: Tasks T_i pushed to the worker's queues

```
1 foreach T_i ∈ LR do
2 |  S_i ← p_i^CPU / p_i^GPU
3 end
4 Sort all ready tasks T_i by decreasing speedup S_i
5 foreach T_i ∈ LR do
6 |  Schedule T_i on the worker w_j achieving the earliest finish time
7 |  push of T_i into queue of worker w_j
8 |  Update processor load time-stamps on worker w_j
9 end
```

time complexity is in $O(n^2 \cdot (m+k))$. It has two major phases: *task prioritizing* and a *worker selection*. Our XKaapi version of HEFT implements both phases during the *activate* operation. The *task prioritizing* phase computes for all ready tasks T_i its speedup S_i relative to an execution on GPU. Next, it sorts the list of ready tasks by decreasing speedups. Whereas the original HEFT rule sorts the tasks by decreasing upward rank (average path length to the end), our rule gives priority on minimizing the sum of the execution times. In the *worker selection* phase, the algorithm selects tasks in the order of their speedup S_i and schedules each task on the worker which minimizes the completion time. Algorithm 1 describes the basic steps of HEFT over XKaapi.

3.2 Dual Approximation and Affinity

Dual Approximation Let us recall first that a ρ-dual approximation scheduling algorithm considers a *guess* λ (which is an estimation of the optimal makespan) and either delivers a schedule of makespan at most $\rho\lambda$ or answers correctly that there exists no schedule of length at most λ [11]. The process is repeated by a classical binary search on λ up to a precision of ϵ. We target $\rho = 2$. The dual approximation part of Algorithm 2 consists in the following steps:

- Choice of the initial guess λ (lines 2 and 4);
- Extract the tasks which fit only into GPUs ($p_i^{CPU} > \lambda$), and symmetrically those which are dedicated to CPUs (line 9);
- Keep this schedule if the tasks fit into λ (line 12). Otherwise, reject it if there is a task larger than λ on both CPUs and GPUs (line 15);
- Add to the tasks allocated to the GPU those which have the largest speedup S_i up to overreaching the threshold λ (line 19) which guarantees the ratio $\rho = 2$;
- Put all the remaining tasks in the m CPUs and execute them using an earliest-finish-time scheduling policy (line 19).

Algorithm 2: DADA – *activate* operation.

Input : A list of ready tasks T_i LR
Output: Tasks T_i pushed to the worker's queues

```
1  lower ← 0
2  upper ← Σ_i max(p_i^CPU, p_i^GPU)
3  while (upper − lower) > ε do
4  |  λ ← (upper + lower)/2
5  |  begin local affinity phase
6  |  |  Schedule tasks of LR per affinity score on its affinity processor, loading
   |  |  each processor up to overreaching αλ
7  |  end
8  |  begin global balance phase
9  |  |  Schedule LR to minimize finish time using λ as hint
10 |  |  if tasks do fit into (2 + α)λ then
11 |  |  |  upper ← λ
12 |  |  |  Keep current schedule
13 |  |  else
14 |  |  |  lower ← λ
15 |  |  |  Reject current schedule
16 |  |  end
17 |  end
18 end
19 Push each task T_i of LR on queue of worker w_j based on the last fitting
   schedule and update processor load time-stamps
```

Affinity DADA builds a compromise taking into account both raw performance and transfers. The principle consists in two successive phases: a first local phase targeting the reduction of the communications through the abstraction described below and a second phase which counter-balances the induced serialization aiming at a global balance. Any algorithm optimizing the makespan could be used for the second phase. We use a basic dual-approximation. In order to gain a finer control, the length of the first phase is controlled by a parameter (denoted by α, $0 \leq \alpha \leq 1$). A value of 0 for α means that the affinity is not taken into account: DADA is then a basic dual-approximation. While at the opposite a value close to 1 allows a length up to λ for the first phase, thus giving a greater weight to affinity.

Each pair (task, computation resource) is given an affinity score. Maximizing the score over the whole schedule enables to consider local impacts. The affinity scores are computed using extra information of the runtime system. In our implementation, they were computed using the amount of data updated by each task. For instance, a task that *writes* or *modifies* a data stored on a resource R has a high score and is prone to be scheduled on R.

4 Experiments

4.1 Experimental setup: Platform & Benchmarks

Platform All experiments have been conducted on a heterogeneous, multi-GPU system. It is composed of two hexa-core Intel Xeon X5650 CPUs running at 2.66 GHz with 72 GB of memory. It is enhanced with eight NVIDIA Tesla C2050 GPUs (Fermi architecture) of 448 GPU cores (scalar processors) running at 1.15 GHz each (2688 GPU cores total) with 3 GB GDDR5 per GPU (18 GB total). The machine has 4 PCIe switches to support up to 8 GPUs. When 2 GPUs share a switch, their aggregated PCIe bandwidth is bounded by the one of a single PCIe 16x. Experiments using up to 4 GPUs avoid this bandwidth constraint by using at most 1 GPU per PCIe switch.

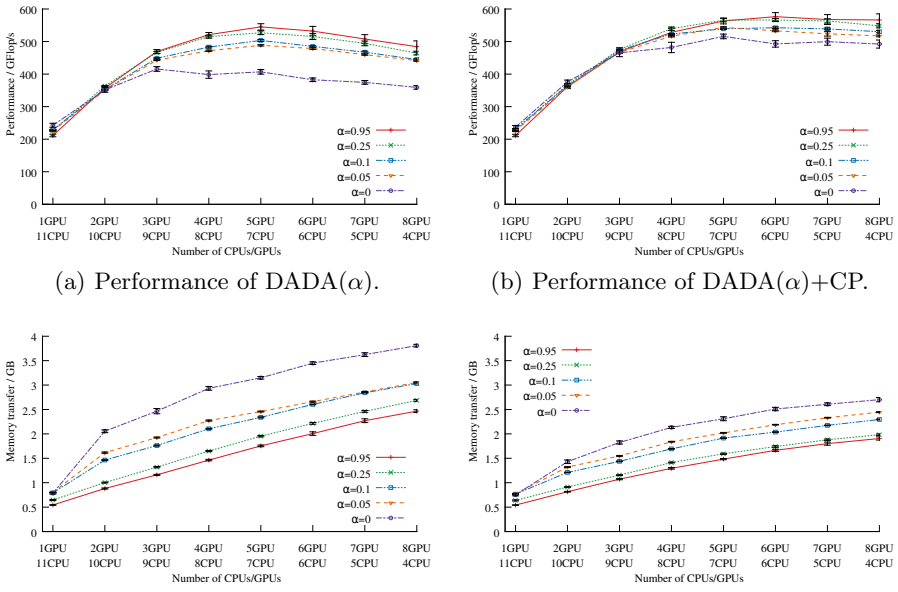
Benchmarks All benchmarks ran on top of a GNU/Linux Debian 6.0.2 *squeeze* with kernel 2.6.32-5-amd64. We compiled with GCC 4.4 and linked against CUDA 5.0 and the library ATLAS 3.9.39 (BLAS and LAPACK). All experiments use the tile algorithms of PLASMA [7] for Cholesky (`DPOTRF`), LU (`DGETRF`), and QR (`DGEQRF`). The QUARK API [17] has been implemented and extended in XKaapi to support task multi-specialization: the XKaapi runtime system maintains the CPU and GPU versions for each PLASMA task. At the task execution, our QUARK version runs the appropriate task implementation in accordance with the worker architecture. The GPU kernels of QR and LU are based on previous works from [1,2] and adapted from PLASMA CPU algorithm and MAGMA from [15]. Each running GPU monopolizes a CPU core to manage its worker. The remaining CPU cores are involved in the application computations.

Methodology Each experiment has been executed at least 30 times for each set of parameters and we report on all the figures (Fig. 1, 2, 3 and 4) the mean and the 95% confidence interval. The factorizations have been done in double precision floating-point operations with a PLASMA internal block (IB) of size 128 and tiles of size 512. For each of them, we plot the highest performance obtained on various matrix sizes with the discussed scheduling strategies.

In the following, DADA(α) represents DADA parametrized by α. We denote by DADA(α)+CP the algorithm using Communication Prediction as supplementary information. HEFT strategy always computes the earliest finish time of a task taking into account the time to transfer data before executing the task.

4.2 Impact of the affinity control parameter α

This section highlights the impact of the affinity control parameter α on the compromise between performance and data transfers. The measures have been done with the Cholesky decomposition on matrices of size 8192×8192 and 16384×16384. However, we present only results for the smallest size as we observe similar behaviors for both matrix sizes.



(a) Performance of DADA(α).

(b) Performance of DADA(α)+CP.

(c) Memory transfer of DADA(α).

(d) Memory transfer of DADA(α)+CP.

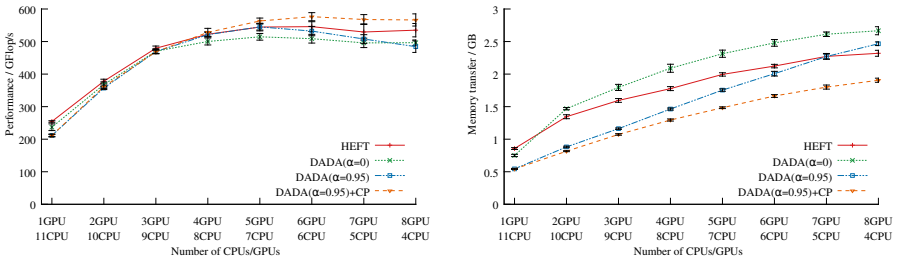
Fig. 1. Impact of parameter α on Cholesky (`DPOTRF`) with matrix of size 8192×8192.

Fig. 1 shows both performance (Fig. 1(a) and 1(b)) and total memory transfers (Fig. 1(c) and 1(d)) for several values of α with respect to the number of GPUs. Both metrics are shown without (Fig. 1(a) and 1(c)) and with (Fig. 1(b) and 1(d)) communication prediction taken into account. Once affinity is considered (*i.e.* $\alpha \neq 0$), the higher the value of α, the better the policy scales. Using as little information as possible (*i.e.* DADA(0) and no communication prediction), the policy performance does not scale with more than two GPUs due to a too huge amount of transfers.

4.3 Comparison of scheduling strategies

We present in this section the results for the three kernels with matrix size 8192×8192. Other tested sizes have the same behavior. The idea is to evaluate the behavior of each strategy with different work loads. Both performance and data transfers of the policies introduced above: HEFT, DADA(0), DADA(α) and DADA(α)+CP are studied.

Experimental evaluation Fig. 2 reports the behavior of the Cholesky decomposition (`DPOTRF`) with respect to the number of GPUs used. It studies both performance results (Fig. 2(a)) and total memory transfers (Fig. 2(b)). All scheduling algorithms have similar performances. DADA(α)+CP scales slightly better with the number of GPU. As expected DADA(α)+CP and DADA(α)



(a) Performance (8192 × 8192). (b) Memory Transfer (8192 × 8192).

Fig. 2. Benchmarks of Cholesky (`DPOTRF`).

are the policies with the lowest bandwidth footprint up to 6 GPU. Yet, as the number of GPU grows, the use of communication prediction allows to reduce the communication volume with sustained high performances.

Fig. 3 reports the behavior of the LU factorization (`DGETRF`). It studies both performance results (Fig. 3(a)) and total memory transfers (Fig. 3(b)). Apart from the performance of DADA+CP for six CPUs and six GPUs (with a large confidence interval), all scheduling policies sustain the same performance. Data transfers seem to have a little impact on performance. However, DADA(α)+CP generates less memory movements than other strategies. DADA(0) is the most costly policy while DADA(α) and HEFT have similar impacts.

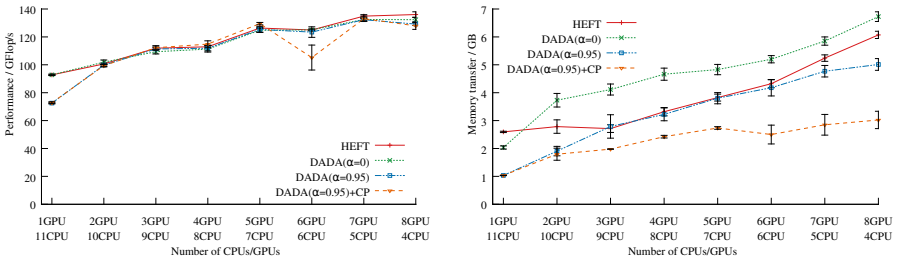
The total memory transfers have the same shape than for the Cholesky factorization. Still, the gap between the curves is widening: DADA(α)+CP is 3.5 less demanding in bandwidth than HEFT for only a slowdown of about 1.13 in performance for 8 GPU.

Finally, Fig. 4 reports the behavior of the QR factorization (`DGEQRF`) with respect to the number of GPUs used. Both performance results (Fig. 4(a)-) and total memory transfers (Fig. 4(b)) are studied. All dual approximations (DADA(0), DADA(α), DADA(α)+CP) behave the same and are outperformed by HEFT. Even the low transfer footprint of both DADA(α) is not able to sustain performance. It seems that the dependencies between tasks for QR factorization have a strong impact on the schedule computed by all dual approximation algorithms. We are still investigating this particular point.

Discussion

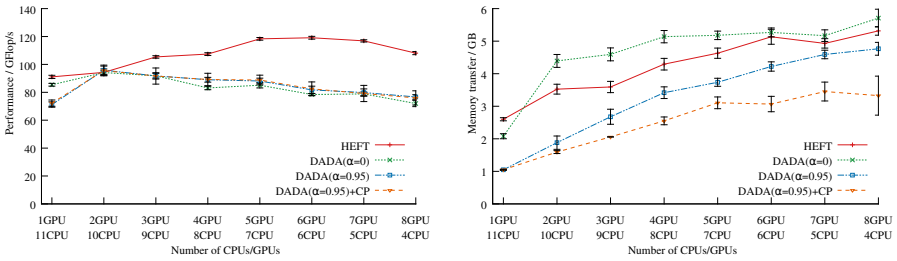
Communication prediction Affinity is a viable alternative to communication modeling. Indeed, DADA without communication prediction is comparable to HEFT in terms of performance. Moreover, affinity based policy combined with communication prediction allows to reduce further more memory transfers (up to a factor 3.5 when compared to HEFT).

Comparison with work stealing scheduling algorithm For the sake of completeness, we also tested the work stealing algorithm. However we did not plot the



(a) Performance (8192 × 8192). (b) Memory Transfer (8192 × 8192).

Fig. 3. Benchmarks of LU (`DGETRF`).



(a) Performance (8192 × 8192). (b) Memory Transfer (8192 × 8192).

Fig. 4. Benchmarks of QR (`DGEQRF`).

results in previous figures for the sake of readability. We briefly discuss them now. The naive work stealing algorithm is cache unfriendly, especially with small matrices as its random choices are heavily penalizing [9]. On the contrary, the affinity policies proposed here are suitable for this case. When scheduling for medium and large matrix sizes, the impact of modeling inaccuracies grows. Model oblivious algorithms such as work-stealing behave well by efficiently overlapping communications and computations while HEFT is induced in error by the imprecise communication prediction. Hence, our approach is much more robust than work stealing and HEFT since it does not need a too precise communication model and adapts well to various matrix sizes.

5 Related Works

StarPU [4], OmpSs [6] and QUARK [17] are programming environments or libraries that enables to automatically schedule tasks with data flow dependencies. OmpSs is based on OpenMP-like pragmas while StarPU and QUARK are C libraries. QUARK does not schedule tasks on multi-GPU architecture and implements a centralized greedy list scheduling algorithm. OmpSs locality-aware scheduling, similar to our data-aware heuristic from [9], computes an affinity

score based on where the data is located and its size. Then, the task is placed on the highest affinity resource or in a global list, otherwise. The StarPU scheduler uses the HEFT [16] algorithm to schedule all ready tasks in accordance with the cost models for data transfer and task execution time [3]. Our data transfer model is based on the StarPU model with minor extension. In the context of dense linear algebra algorithms, PLASMA [7] provides fine-grained parallel linear algebra routines with dynamic scheduling through QUARK, which was conceived specially for numerical algorithms on multi-CPU architecture. MAGMA [15] implements static scheduling for linear algebra algorithms on heterogeneous systems composed of GPUs. Recently it has included some methods with dynamic scheduling in multi-CPU and multi-GPU sytems on top of StarPU, in addition to the static multi-GPU version. In [14] the authors based their Cholesky factorization on 2D block cyclic distribution with an owner compute rule to map tasks to resources. DAGuE [5] is a parallel framework focused on multi-core clusters and supports single-GPU nodes. Other papers reported performance results of task-based algorithms with HEFT cost model scheduling on heterogeneous architectures for the Cholesky [4], LU [1], and QR [2] factorizations. All the results report evaluation of single floating point arithmetics with up to 3 GPUs. Due to the small number of GPUs, such studies cannot observe contention and scalability.

6 Conclusion

We presented in this paper a new scheduling algorithm on top of the XKaapi runtime system. It is based on a dual approximation scheme with affinity and has been compared to the classical HEFT algorithm for three tile algorithms from PLASMA on an heterogeneous architecture composed of 8 GPUs and 12 CPUs. Both algorithms attained significant speed up on the three dense linear algebra kernel. Moreover, if HEFT achieves the best absolute performance with respect to DADA on QR, while DADA has similar or better performances than HEFT on Cholesky and LU for large numbers of GPU. Nevertheless, DADA allows to significantly reduce the data transfers with respect to HEFT. More interesting, thanks to its affinity criteria DADA can introduce communication in the scheduling without too precise communication cost model which are required in HEFT to predict the completion time of tasks.

We would like to extend the experimental evaluations on robustness of scheduling with respect to uncertainties in cost models, especially on the communication cost which is very sensitive to contentions that may appear at runtime. Another interesting issue would be to study other affinity functions.

Acknowledgments

This work has been partially supported by the French Ministry of Defense – DGA, the ANR 09-COSI-011-05 Project Repdyn and CAPES/Brazil.

References

1. Agullo, E., Augonnet, C., Dongarra, J., Faverge, M., Langou, J., Ltaief, H., Tomov, S.: Lu factorization for accelerator-based systems. In: IEEE/ACS AICCSA. pp. 217–224. AICCSA '11, IEEE Computer Society, Washington, DC, USA (2011)
2. Agullo, E., Augonnet, C., Dongarra, J., Faverge, M., Ltaief, H., Thibault, S., Tomov, S.: QR Factorization on a Multicore Node Enhanced with Multiple GPU Accelerators. In: IEEE IPDPS. EUA (2011)
3. Augonnet, C., Thibault, S., Namyst, R.: Automatic calibration of performance models on heterogeneous multicore architectures. In: Euro-Par. pp. 56–65. Springer-Verlag (2010)
4. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice and Experience 23(2), 187–198 (2011)
5. Bosilca, G., Bouteiller, A., Danalis, A., Herault, T., Lemarinier, P., Dongarra, J.: DAGuE: A generic distributed DAG engine for High Performance Computing. Parallel Computing 38(1–2), 37–51 (2012)
6. Bueno, J., Planas, J., Duran, A., Badia, R.M., Martorell, X., Ayguadé, E., Labarta, J.: Productive Programming of GPU Clusters with OmpSs. In: IEEE IPDPS (2012)
7. Buttari, A., Langou, J., Kurzak, J., Dongarra, J.: A class of parallel tiled linear algebra algorithms for multicore architectures. Parallel Computing 35(1), 38–53 (2009)
8. Gautier, T., Besseron, X., Pigeon, L.: KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In: PASCO'07. ACM, London, Canada (2007)
9. Gautier, T., Lima, J.V., Maillard, N., Raffin, B.: XKaapi: A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures. In: IEEE IPDPS. pp. 1299–1308 (2013)
10. Hermann, E., Raffin, B., Faure, F., Gautier, T., Allard, J.: Multi-GPU and Multi-CPU Parallelization for Interactive Physics Simulations. In: Euro-Par. vol. 6272, pp. 235–246. Springer (2010)
11. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. J. ACM 34(1), 144–162 (Jan 1987)
12. Kedad-Sidhoum, S., Monna, F., Mounié, G., Trystram, D.: Scheduling independent tasks on multi-cores with gpu accelerators. In: 11th HeteroPar Workshop (2013)
13. Lima, J.V.F., Gautier, T., Maillard, N., Danjean, V.: Exploiting Concurrent GPU Operations for Efficient Work Stealing on Multi-GPUs. In: 24th SBAC-PAD. pp. 75–82. IEEE, New York, USA (2012)
14. Song, F., Dongarra, J.: A scalable framework for heterogeneous GPU-based clusters. In: ACM SPAA. pp. 91–100. ACM, New York, NY, USA (2012)
15. Tomov, S., Dongarra, J., Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Computing 36(5-6), 232–240 (2010)
16. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE TPDC 13(3), 260–274 (2002)
17. YarKhan, A., Kurzak, J., Dongarra, J.: Quark users' guide: Queueing and runtime for kernels. Tech. Rep. ICL-UT-11-02, University of Tennessee (2011)